PTO Customer Number 22883

Title:	File Access Control in a Multi-Protocol File Server
Inventor:	Hitz
Docket:	103.1019.10

TECHNICAL APPENDIX

6 Total Pages

(not including this cover page)

COVER PAGE ONLY – NOT PART OF TECHNICAL APPENDIX

[NO PAGE NUMBER]

File System Security: Secure Network Data Sharing for NT and UNIX

1. Introduction

This paper describes a merged file system security model for NT and UNIX. The merged security model was designed to meet several goals:

(1) Make Windows and NT Users Happy.

Support an NT-centric security model based on NT Access Control Lists (ACLs). To a Win95 or NT client using CIFS (the standard Windows file service protocol) this security model should behave exactly like an NT file server.

(2) Make UNIX Users Happy.

Support a UNIX-centric security model based on UNIX permissions. To an NFS client, this security model should behave exactly like a UNIX NFS server. (This is the only security model that filers have historically supported.)

(3) Let Them Work Together.

Provide reasonable heuristics so that Windows users can access files with UNIX-centric security, and UNIX users can access NT-centric files.

To meet these goals, NetApp supports both NTFS-trees and UNIX-trees. Administrators can set the security model on the root of a WAFL file system, but they can also set the security for individual quota trees, allowing Windows and UNIX users both to be happy in the same file system.

Native requests -- NFS to a UNIX-tree or CIFS to an NTFS-tree -- work exactly as expected. UNIX-trees are modeled after Solaris, and NTFS-trees are modeled after NT. Non-native requests use heuristics designed to operate as intuitively as possible while still maintaining security.

CIFS requests to UNIX-trees are handled by mapping each CIFS user to an equivalent UNIX user, and then validating against the standard UNIX permissions.

In the simplest case, a user named John might have the account "john" on both NT and UNIX. When John starts a CIFS session, the filer looks up "john" in /etc/passwd (or over NIS), and uses the specified UID and GID for all access validation.
A user name mapping file handles the case where John's NT account is "john", but his UNIX account is "jsmith". NT users with no UNIX account may be mapped to a specified UNIX account, or they may simply be denied access.

NFS requests to NTFS-trees are validated using special UNIX permissions that are set whenever an ACL is updated. The UNIX permissions are guaranteed to be more restrictive than the ACL, which means that users can never circumvent ACL-based security by coming in through NFS. On the other hand, since UNIX permissions are less rich than NT ACLs, a multiprotocol user may be unable to access some files over NFS even though they are accessible over CIFS. In practice, NFS access to NTFS-trees works well for the owner, and for access granted to the NT "everyone" account, but not for other cases.

In addition to UNIX-trees and NTFS-trees, the filer supports a Mixed-tree in which the security model is determined on a file-by-file basis. Files created by UNIX users use UNIX permissions, and files created by NT users use NT ACLs. A file's security model may be flipped from one style to another by NFS "setattribute" or CIFS "set ACL" requests. Only a file's owner can flip the style.

- How are requests to display permissions handled?
- How are requests to set permissions handled?
- How are permissions set for newly created files?

Handling non-native permissions is tricky, because the NT and UNIX permission models are so different. To define a merged security model, one must specify how an NT ACL will be converted to UNIX permissions, and visa-versa.

3. Handling Non-Native CIFS Requests

This section describes how we handle CIFS requests to UNIX-style files. Remember, files with UNIX permissions occur both in UNIX-trees, where all files are UNIX-style, and in Mixed-trees, which have both UNIX-style and NTFS-style files.

Section 3.1 discusses how non-native CIFS requests are validated, and section 3.2 discusses the non-native handling for displaying permissions, setting permissions, and creating new files.

3.1 Validating Non-Native CIFS Requests

NetApp filers validate CIFS requests to UNIX-style files by generating a mapped UID (and GIDs) for each CIFS session, and then using the UID (and GIDs) to check against the UNIX permissions.

Suppose that the NT user "john" connects to a filer. Here are the steps that the filer will take to determine the mapped UID and GIDs for "john".

(1) The filer sends a request to the NT domain controller (DC), to authenticate "john", and to find out the NT SID for "john".

(2) The filer looks in the user mapping file to determine whether the NT account "john" maps into a different account name under UNIX. In this case, let's assume that "john" maps into the UNIX account "jsmith".

(3) The filer looks up "jsmith" in /etc/passwd (possibly via NIS) to determine the UNIX UID for John.

(4) The filer uses /etc/groups (possibly via NIS) to determine the UNIX GIDs for John.

These steps provide each CIFS session with a full set of UNIX authentication information, which allows the filer to easily validate most requests against the UNIX permissions.

Some CIFS operations don't map well to UNIX operations, so they must be handled specially:

- Set ACL

The CIFS "set ACL" operation is always denied in UNIX-trees. In Mixed-trees, a "set ACL" operation is only allowed by the owner (i.e. the mapped UID for the CIFS session must match the file's UID.) In this case, the file is converted from UNIX-style permissions to NT-style permissions.

Only the owner can set an ACL, because in UNIX only the owner is allowed to set attributes.

- Take Ownership

The CIFS "take ownership" operation is always denied in UNIX-trees. In Mixed-trees, only the file's owner can "take ownership". Like the "set ACL" request, this converts the file to NT-style permissions.

3.2 Request Processing for Non-Native CIFS Requests

This section considers non-native CIFS requests in light of the three questions listed above, in Section 2.2, "Important Issues for Non-Native Filesystem Security":

- How are requests to display permissions handled?
- How are requests to set permissions handled?
- How are requests to create a file handled?

3.2.1 How are requests to display permissions handled?

For non-native CIFS requests to display permissions, WAFL dynamically builds an ACL designed to represent the UNIX permission as best as possible.

On might hope to build an NT ACL that perfectly represents the UNIX permission like this:

Owner - map the file's UID into an NT SID.
Group - map the file's GID into an NT SID.
ACL
ACE for owner SID, based on UNIX user perms.
ACE for group SID, based on UNIX group perms.
ACE for well known NT "everyone" SID, based on UNIX other perms.

In practice, this doesn't work because we have no way to convert UIDs or GIDs into SIDs. In constructing the ACL, we can only use well-known NT SIDs, and the SID for the CIFS session itself. These are sufficient to let us construct an ACL that, while not perfect, does provide useful information.

Each ACL contains two access control entries (ACEs):

- An entry for NT "everyone" SID, based on the UNIX "other" permissions.

- An entry for the SID of the CIFS session, based on whichever UNIX permission is appropriate. If the mapped UID is the file's owner, the ACE is based on the UNIX owner perms. If the group matches, then it's based on the group perms. Otherwise it's based on the other perms.

If the CIFS session owns the file, then in the faked up ACL the session's SID is shown as the owner. If not, then the well known NT SID "CREATOR_OWNER" is shown as the owner.

3.2.2 How are requests to set permissions handled?

In UNIX-trees, CIFS requests to set permissions are always rejected. Outside of UNIX-trees, non-native requests to set permissions are allowed only by the file's owner. If allowed, the ACL takes effect just as it would have if the file had been an NT-style file. After the set ACL request is processed, the file becomes an NT-style file.

3.2.3 How are permissions set for newly created files?

Unlike UNIX, which passes the permissions for the new file as part of the create request, NT expects permissions to be inherited from the parent directory.

To handle CIFS create requests in a UNIX-style tree, the filer simply inherits the parent directory's UNIX permissions:

- The file's owner is set to the mapped UID fo the CIFS session.

- The file's group is set to the mapped GID for the CIFS session.

- The permission bits are inherited directly from the parent directory, except that SUID and SGID bits are cleared for non-directory creates.

4. Handling Non-Native NFS Requests

This section describes how we handle NFS requests to NTFS-style files -- that is files with NT ACLs.

Section 4.1 discusses how non-native CIFS requests are validated, and section 4.2 discusses the non-native handling for displaying permissions, setting permissions, and creating new files.

4.1 Validating Non-Native NFS Requests

Whenever an NT ACL is set or changed, WAFL calculates a corresponding set of UNIX permissions. As a result, very little special processing is required to validate NFS requests to files with NT ACLs. Simply doing the normal checks against the UNIX permissions does (almost) the right thing. The rest of this section describes how the UNIX permissions are constructed from the ACL, and explains the "(almost)" in the previous sentence.

Converting an NT ACL into UNIX permissions is surprisingly tricky. This section gives a brief overview, but an observant user may encounter slight differences in the actual implementation.

- The file's UID is set to the mapped UID for the CIFS session.

- The file's GID is set to the mapped GID for the CIFS session.

- The UNIX user perm is set based on the access rights that the ACL grants to the CIFS session creating the file. (These may or may not be explicitly specified by an ACE for the owner.)

- The UNIX other perm is set based on the access rights granted to the NT "everyone" account. (If the ACL contains any denies, then the denied permissions are subtracted from the other perms.)

- The UNIX group perm is set equal to the other perm.

This design avoids security holes by ensuring that the UNIX permission is always at least as restrictive as the NT ACL. Unfortunately, UNIX permissions cannot represent the full richness of the NT security model. As a result, a file that a user can reach via CIFS may not be accessible via NFS.

Because NT supports some specific permissions that UNIX lacks, it is not possible to rely entirely on the UNIX permissions to validate NFS requests:

REMOVE/RMDIR
Only the owner of a file is allowed to delete it. This is necessary to avoid violating the NT "delete child" permission.

CREATE
Only the owner of a directory can create anything in it. This is required in order for NT ACL inheritance to work properly.

4.2 Request Processing for Non-Native NFS Requests

This section considers non-native NFS requests in light of the three questions listed above, in Section 2.2, "Important Issues for Non-Native Filesystem Security":

- How are requests to display permissions handled?
- How are requests to set permissions handled?
- How are permissions set for newly created files?

4.2.1 How are requests to display permissions handled?

As described above, in section 4.1, every file with an NT ACL also has a set of UNIX permissions stored with it. To handle an NFS "get attributes" request, the filer simply returns those stored permissions.

4.2.2 How are requests to set permissions handled?

In NTFS-trees, NFS requests to set permissions are always rejected.

In Mixed-qtrees, only a file's owner is allowed to set permissions. When UNIX permissions are set on a file, the NT ACL is deleted; the file changes from NT-style to UNIX-style.

Note that "set attribute" requests that update non-security information such as access time or modify time are allowed even in NTFS-trees, and they do not delete the NT ACL.

4.2.3 How are permissions set for newly created files?

NFS creates in NTFS-trees are only allowed by the directory's owner. This is because an NT SID is required to handle NT ACL inheritance.
An NFS request has no SID, but for a create request from a directory's owner, WAFL can use the owning SID from the directory's ACL and handle ACL inheritance according to normal NT rules. (If the parent directory has no ACL, WAFL follows normal UNIX rules.)

In Mixed-trees, NFS create requests are handled according to normal UNIX rules.

5. Other Issues

5.1 FAT versus NTFS

NT servers support both FAT file systems and NTFS file systems. FAT is the traditional DOS file system -- it has no file-level security at all. The NTFS file system was designed for NT, and it supports a security model based on ACLs.

Since NTFS-trees and Mixed-trees both support ACLs, they must be advertised as "NTFS" file systems. (If they were advertised as "FAT", clients would assume that they had no ACLs, and would disable the interfaces for controlling ACLs.)

It is less obvious how to advertise UNIX-trees. One can make a case either way:

- UNIX-trees don't support ACLs, so advertising them as FAT sends a clear message to clients not to try to use ACLs. Advertising as NTFS would be confusing, since no ACLs are really present and any request to set ACLs will fail.

- UNIX-trees support file level security, and advertizing them as NTFS allows the filer to display the UNIX permissions using faked-up ACLs. Advertizing as FAT would be confusing, because it would seem to imply that no file-level security is present.

In the end we decided to advertize UNIX-trees as FAT, because this seems least likely to confuse Windows programs that absolutely must have ACLs.

Still, there are several situations in which it is useful to construct a fake ACL for an NT-style file, as described above in 3.2.1:

(1) In Mixed-trees

Mixed-trees contain files both UNIX-style and NTFS-style files. To support ACLs they must be advertized as "NTFS", yet not all files in them contain ACLs.

(2) In NTFS-trees that originated as UNIX or Mixed-trees

A quota tree's security style can be changed at any time, so a tree that began as UNIX-style may later be converted to NTFS. In this case, it will clearly be advertized as "NTFS", but it may contain files without ACLs.

(3) In UNIX-trees accessed via an NTFS or Mixed Share.

The root of a filesystem may have NTFS or Mixed security, but it may contain a UNIX quota tree. In this case, the C$ (or root) share will be advertized as "NTFS", but there is nothing to stop a user from going down into the UNIX quota tree and trying to display an ACL.

5.2 Migration

For sites already using filers with CIFS, migration to the NTFS security model is an important issue.

System administrators can update the security model for any quota tree (including the root of the filesystem), using the quota command. The syntax is:

quota qtree_security <quota tree> [unix|ntfs|mixed]

When a UNIX-tree is converted to an NTFS-tree, shares are advertised as "NTFS" instead of "FAT", and ACLs will be dynamically created for the files as described above in section 3. In addition, NetApp will ship a Windows utility to run through a tree and set a real ACL on each file based on its UNIX permissions. This is useful since the dynamically constructed ACL can't show the exact permissions for a file.

When an NTFS-tree is converted to a UNIX-tree, shares are advertized as "FAT" instead of "NTFS", and any ACLs in the tree are simply ignored. ACLs are not actually deleted -- however -- so if the tree is converted back to NTFS, the ACLs will still be present. The best way to delete the ACLs is to write a script that runs as root and chowns each file to it's existing owner. (Remember that doing a chown or chmod deletes the ACL on a file.)

5.3 Share Level ACLs

Share level ACLs in the new release are now based on NT SIDs, and they can be edited over the network using the NT Server Manager.

For backward compatibility, share level ACLs based on UNIX user names will continue to function, although they cannot be controlled via Server Manager.

References:

1. Rob Reichel , "Inside Windows NT Security" , *Windows/DOS Developer's Journal.* April & May 1993.
2. Stephen Sutton, "Windows NT Security Guide". ISBN 0201419696